

High-performance FPGA implementation of the secure hash algorithm 3 for single and multi-message processing

Fatimazahraa Assad¹, Mohamed Fettach¹, Fadwa El Otmani¹, Abderrahim Tragha²

¹Department of Physics, Information Processing Laboratory, Hassan II -Casablanca University, Casablanca, Morocco

²Department of Mathematics, Information Treatment and Modeling Laboratory, Hassan II -Casablanca University, Casablanca, Morocco

Article Info

Article history:

Received Dec 30, 2020

Revised Aug 6, 2021

Accepted Sep 4, 2021

Keywords:

FPGA

Hardware implementation

High performance

High throughput

Keccak

Pipelining

SHA-3

ABSTRACT

The secure hash function has become the default choice for information security, especially in applications that require data storing or manipulation. Consequently, optimized implementations of these functions in terms of Throughput or Area are in high demand. In this work we propose a new conception of the secure hash algorithm 3 (SHA-3), which aim to increase the performance of this function by using pipelining, four types of pipelining are proposed two, three, four, and six pipelining stages. This approach allows us to design data paths of SHA-3 with higher Throughput and higher clock frequencies. The design reaches a maximum Throughput of 102.98 Gbps on Virtex 5 and 115.124 Gbps on Virtex 6 in the case of the 6 stages, for 512 bits output length. Although the utilization of the resource increase with the increase of the number of the cores used in each one of the cases. The proposed designs are coded in very high-speed integrated circuits program (VHSIC) hardware description language (VHDL) and implemented in Xilinx Virtex-5 and Virtex-6 A field-programmable gate array (FPGA) devices and compared to existing FPGA implementations.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Fatimazahraa Assad

Department of Physics, Information Processing Laboratory, Hassan II -Casablanca University

Casablanca B.P 7955, Morocco

Email: fatimazahraa.assad@gmail.com

1. INTRODUCTION

Due to the attacks on the message-digest algorithm 5 (MD5) and secure hash algorithm (SHA-1) hash algorithms [1], [2] the National Institute of Standards and Technology (NIST) has organized a public competition to develop a new hash standard, in which 64 candidates' algorithms were submitted to NIST for consideration. Among these 51 met the minimum acceptance criteria marking the beginning of the first round of the SHA-3 Competition. The selection of the candidates was based on various evaluation criteria, only 14 were chosen for the second round of the competition. Keccak has been selected as the SHA-3 standard [3]. Keccak is based on a sponge construction which differs from the Merkle–Damgård construction used by MD5, SHA1, and SHA-3 [4]–[6]. With this structure, Keccak is more secure than the existing standards [7]. During and after the NIST's competition several implementations of SHA-3 are presented [8]–[17], [18]. In spite of the availability of software implementations of security algorithms using high powerful central processing unit (CPUs) [19], hardware implementations of these algorithms using high developed platforms such as field-programmable gate arrays (FPGAs) and application specific integrated circuit (ASICs) remains greatly demanded.

Optimized hardware implementations of these algorithms in terms of area or throughput are in high demand, depending on whether it concerns slow or fast applications, consequently, these implementations can be optimized either in terms of low area (lightweight implementations) or higher throughput (high-performance

implementation). The first category of these designs is optimized for compact designs by merging Rho, Pi, and Chi steps of the algorithm into a single step [12]–[16]. In the same line of thought, Jungk and Stöttinger [20] have decreased the area cost by using shift register primitives of the FPGA instead of the distributed random access memory (RAM). These algorithms can also be optimized for high Throughput, by applying several optimization techniques such as loop unrolling [16] and pipelining [11]–[20], [21].

In the case of the SHA-3 hash algorithm, the pipeline technique can be employed in SHA-3 in two different manners. An internal pipelining or sub-pipelining, where the registers are introduced inside the hash compression function, Athanasiou *et al.* [11] introduce a pipeline stage between the π and the χ sub-rounds of the algorithm while Mestiri *et al.* [14] inserted two registers between the sub-functions: the first register is inserted between the Pi and Chi, and the second register is implemented at the end of the Keccak round. The principal disadvantage of this type of pipelining lies in the fact that the transformations of the next block cannot start before the output of the previous block becomes available. The second plausible means of pipelining are external pipelining. It's done by placing pipeline registers between two successive rounds, where each stage contains the entire compression function [15].

Presently, cryptographic hash function plays a critical role in many recent fields such as radiofrequency identification (RFID) [22], internet of things (IoT) [23]–[26], medicine [27], wireless sensor network (WSN) [28], [29], and cloud computing [30], [31]. Consequently, the enhancement of the speed/performance of the SHA-3 in hardware platforms is in great demand. e.g., in the IoT where billions of objects need to be connected, data integrity is an indispensable factor. Hardware implementations of the cryptographic hash functions are greatly demanded, so as to secure the communication process between the interconnected objects constituting the IoT applications, mainly in a public network.

In this study, our main focus is increasing the performance of the SHA-3, by applying the pipelining. We have proposed four different pipelined designs, two, three, four, and six-stage pipelines. These designs are capable of handling multi-block messages and processing multiple messages. The remainder of this paper is organized as follows. The second section briefly introduces the SHA-3 specifications; section 3 presents the proposed designs of the SHA-3 hash function; the FPGA synthesis results and comparisons with previously published works are provided in section 4; while the paper's conclusions are discussed in the last section.

2. SHA-3 SPECIFICATIONS

The SHA-3 algorithm, initially known as Keccak, is a cryptographic hash function designed by Bertoni *et al.* [7] unlike SHA-1 [5] and SHA-2 [6], SHA-3 does not depend on the Merkle–Damgard construction. The SHA-3 depends on sponge construction. The purpose of using this structure is its security against generic attacks. Moreover, this construction is characterized by its simplicity and adaptability.

The sponge construction is based on two phases, the first one is the absorption phase where the state, a matrix of SHA-3 composed of 5x5 matrixes of 64-bit words, receives the null matrix then it will be XOR-Ed with the first r-bit message block and the state will be regenerated. This process continues until all the blocks have been retained. The second phase is the squeezing phase, where the output hash value is truncated from the first r-bit, and further changes are made if the required output bit is not acquired.

For the two phases, the same function f is being utilized. Figure 1 demonstrates how the sponge construction absorbs the message blocks M and how the outputs Z are generated. The sponge construction permits arbitrary-length outputs.

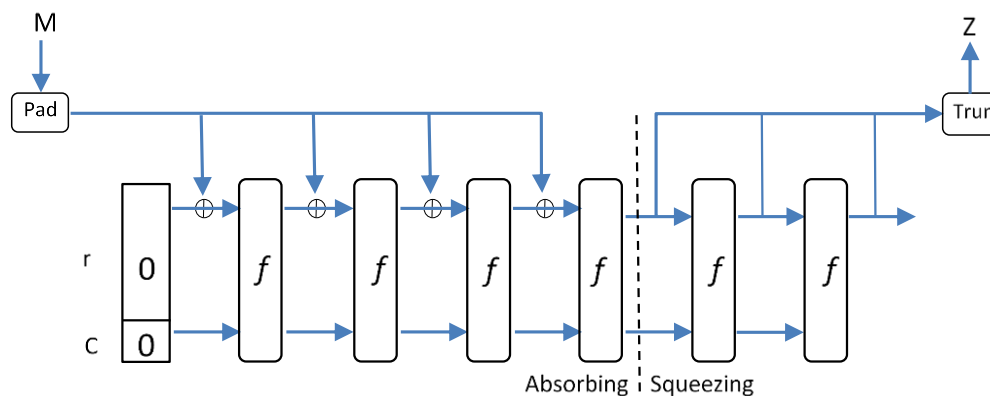


Figure 1. Absorbing and squeezing phases of the sponge construction

The SHA-3 family consists of four cryptographic hash functions, SHA3-224, SHA3-256, SHA3-384, and SHA3-512 [3]. These functions share the same structure, namely, the sponge construction. Depending on the desired output length, two parameters are used for the sponge construction. These two input parameters are the bitrate r or the length of one message block and C is the capacity where $r+C=b$, these parameters can be chosen by the user. The security of the SHA-3 algorithm can be relatively enhanced by increasing the capacity C and reducing the bitrate r as shown in Table 1.

For our design we have taken into consideration the 4 cases, depending on the desired output length. It should be noted that the security of the hashed message can be varied by changing the hash result length. Keccak comprises 24 rounds, each round is sub-divided into five steps: θ (Theta), ρ (Rho), π (Pi), χ (Chi) and ι (Iota) as shown in Figure 2

Table 1. The capacity-bitrate values

Algorithm	r (bits)	C (bits)	Hash output (bits)
SHA-224	1152	448	224
SHA-256	1088	512	256
SHA-384	832	768	384
SHA-512	576	1024	512

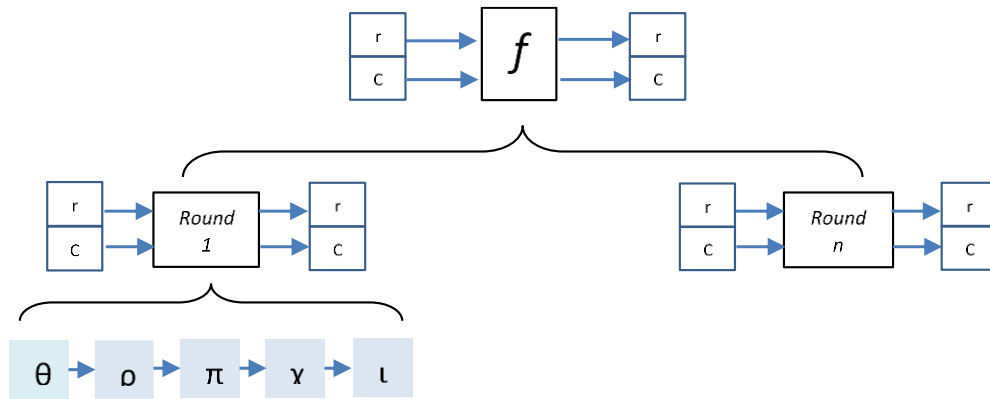


Figure 2. The inner construction of Keccak-f

The mathematical formulas of the five sub-functions (θ , ρ , π , χ , and τ) are presented:

Theta (Θ) step:

$$C(x, z) = \text{Theta_in}(x, 0, z) \oplus \text{Theta_in}(x, 1, z) \oplus \text{Theta_in}(x, 2, z) \oplus \text{Theta_in}(x, 3, z) \oplus \text{Theta_in}(x, 4, z)$$

$$D(x, z) = C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w]$$

$$\text{Theta_out}(x, y, z) = \text{Theta_in}(x, y, z) \oplus D(x, z)$$

Rho (ρ) step:

$$\text{Rho_out}(0, 0, z) = \text{Rho_in}(0, 0, z)$$

$$\text{Let } (x, y) = (1, 0)$$

$$\text{Rho_out}(x, y, z) = \text{Rho_in}[x, y, (z - (t+1)(t+2) / 2) \bmod w]$$

$$(x, y) = (y, (2x + 3y) \bmod 5)$$

Pi (π) step:

$$\text{Pi_out}(x, y, z) = \text{Pi_in}[(x+3y) \bmod 5, x, z]$$

Chi (χ) step:

$$\text{Chi_out}(x, y, z) = \text{Chi_in}(x, y, z) \oplus [\text{NOT}[\text{Chi_in}((x+1) \bmod 5, y, z)]$$

$$\text{AND Chi_in}((x+2) \bmod 5, y, z)]$$

Iota (i) step:

$$\text{Iota_out}(0, 0, z) = \text{Iota_in}(0, 0, z) \oplus \text{RC}(i)$$

RC is a 64-bit value prefixed for each round. The Keccak-f permutation comprises 24 rounds, which are indistinguishable except for the addition of a round-dependent constant. The constants $\text{RC}[i]$ are the round constants, where i is the round's number.

3. RESEARCH METHOD

The basic hardware architecture of the SHA-3 hash function has a 64-bit data input and 4 inputs that control the overall. Thus, a 512-bit (384-bit, 256-bit, or 128 bit) data output and 1-bit output indicates that the hash is ready. The message block has been treated from the beginning until the generation of the hash value, including the padding Unit. Moreover, it is not based on any FPGA resources such as Digital signal processors (DSPs). As appeared in Figure 3 we applied four types of padding taken into account all different output lengths (128, 256, 384, and 512) to build a complete design of the Keccak hash algorithm. The SE signal allows selecting the desired output length.

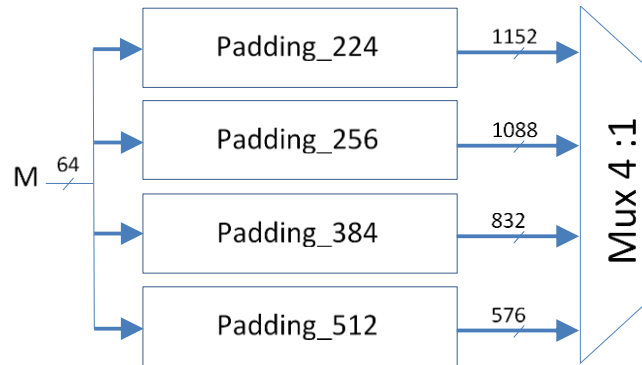


Figure 3. The process of padding

After the process of padding, the padded message must be cut into blocks of the same size, the formula (1) is used to get 1600 bit data, that will be converted subsequently to a state, $data = (\text{Block} \text{ xor } r) // C$. Finally, the 1600-bit data will be converted into three-dimensional arrays (state) by applying in (1).

$$\text{State}(x, y, z) = f(z + 64 * (5 * y + x)) \quad (1)$$

As forced by the algorithm, during the first iteration an initial zero-state is utilized (Bitrate r_0 and capacity C_0 .) [7]. The state of SHA-3 constitutes 5x5 matrixes of 64-bit words, as appeared in Figure 4 with: $0 \leq x \leq 4$; $0 \leq y \leq 4$; $0 \leq z \leq 63$.

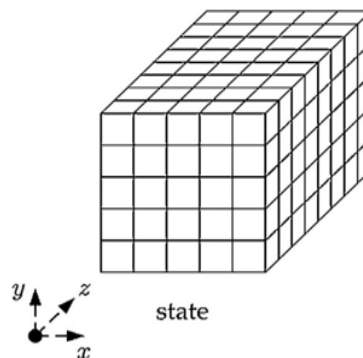


Figure 4. The state of SHA-3

At the beginning of the round, a two-to-one multiplexer is used, this Multiplexer drives the data from the preprocessing block to the main component of SHA-3. The principal functionality of this Mux allows processing the twenty-four rounds. When *Ctrl_Mux* is low, the multiplexer selects the input data and at high, the Mux will select the feedback data. In other words, the input data are selected for the first round and the feedback data for the other twenty-three rounds.

The output of the Mux is forwarded to the compression functions which are the main unit of SHA-3. During this step, the computation part of the algorithm will be done. This module consists of 24 processing rounds of five permutations (θ , ρ , π , χ , and τ). The mathematical formulas of these permutations are given above. At the end of the round, a register is used in order to synchronize the data path, the utility of this register is to store the result of each round. Then a 1×2 demultiplexer is used to process the twenty-four rounds of the compression function: i) the demultiplexer receives the feedback data when *Ctrl_DMux* is low else, ii) the DMux will select the input of truncating block.

The hash value is generated after 24 iterations, so 24 constant values are needed to compute the hash result. In our proposed architecture, the appropriate round constants (*RC*) that are used in the last sub-function Iota are selected using a counter (0 to 23) generated by the control unit which this counter is the address bus of the ROM where we stored the *RC* constants defined by the algorithm [7]. Finally, after twenty-four rounds the output is simply truncated for producing the final hash value with the desired length. It should be noted that before generating the final hash value, it should apply the inversion procedure of mapping to get the right output. The *HR* signal indicates that the hash result is ready.

As mentioned previously the design is controlled using the control unit, which is implemented using an FSM. This FSM is used to control and synchronize the different components of the SHA-3 design. The control unit has 3 inputs *ST*, *Reset*, and *Clk*, three 1-bit outputs *Ctrl_Mux*, *Ctrl_DMux*, *HR*, and a 5-bit output *CMPT*. The FSM includes three states, state *S1* or the initial state is for the preprocessing of the input data, state *S2* for the main computation thus it includes a counter that counts up to 24, which corresponds to 24 iterations of the transformation round. *S3* is used for the truncating of the message block and indicates that the hash result is ready by giving '1' to the *HR* signal. The design needs one clock cycle to operate one round. Consequently, 24 clock cycles are needed to compute 24 iterations. The hash value will be ready after twenty-four clock cycles.

Based on (2), the *Throughput* of the SHA-3 can be improved in two ways, either increasing the numerator presented by *Frequency* or decreasing the denominator (clock cycles). In this work, we opted for the second possibility by using the pipelining technique. The effect of using pipelining is to reduce the number of clock cycles required in each round, which implies increasing the *Throughput* according to (2). Four different designs are proposed two, three, four, and six-stage pipelines. Beginning with the case of two-stage pipelines. The design Figure 5 permits yielding the first block after 24 cycles and delivers a new block every 12 cycles. This allowed processing multiple messages for hashing simultaneously. This core contains the implementation of the five sub-functions and the output of this component will be stored in a register. The output of the register is fed as input to the following stage multiplexer. At the last cycle, the output is fed to the truncate unit to produce the final hash value. The 5-bit counter will be replaced by a 4-bit counter that steers the twelve iterations at each core. It should be mentioned that the size of the ROMs used to store the constant values will be adjusted according to the number of cores used in each case. In the case of the two-stage pipeline, two cores are needed instead of a single core. Accordingly, the size of the constant unit is adjusted Figure 5.

To describe the pipelining mechanism, assuming that we have 3 messages of one block or more denoted B_{mn} , the block n of the message m , during the first 12 clock cycles the block B_{11} will be processed by the first stage, and the second stage is empty. Then, during the last 12 clock cycles, B_{11} will be processed by the 2nd stage and stage 1 receives block B_{21} . In the same way, the blocks B_{12} , B_{22} , B_{13} , B_{23} , ... will be processed by the 2 stages. If there are other blocks of the first 2 messages, they must be processed otherwise, it is the turn of the first block of the 3rd message B_{31} to be hashed and so on. At the last cycle, the output is fed to the truncating unit to be truncated and produce the final hash value.

The three-stage pipeline design allows outputting the first block after exactly 24 clock cycles and can process a new block every 8 clock cycles instead of 12 in the case of pipelining 2 stages. The 4-bit counter will be replaced by a 3-bit counter that steers the six iterations at each core of the three cores and the appropriate round constants. The four-stage pipeline design allows the output of the first block after exactly 24 clock cycles and can process a new block (receive the block and compute its hash value) every 6 clock cycles. In the case of the four-stage pipeline, it is necessary to also replace the 4-bit counter with a 3-bit one so as to count up to 6. The case of a six-stage pipeline that permits the generation of the first block after 24 cycles and delivers a new block every 4 cycles as depicted in Figure 6. To ensure the good functioning of the system, the 3-bit counter will be replaced by a 2-bit counter. Six counters are needed in this case as appeared in Figure 6. Also, the size of the ROMs used to store the constant values will be adjusted.

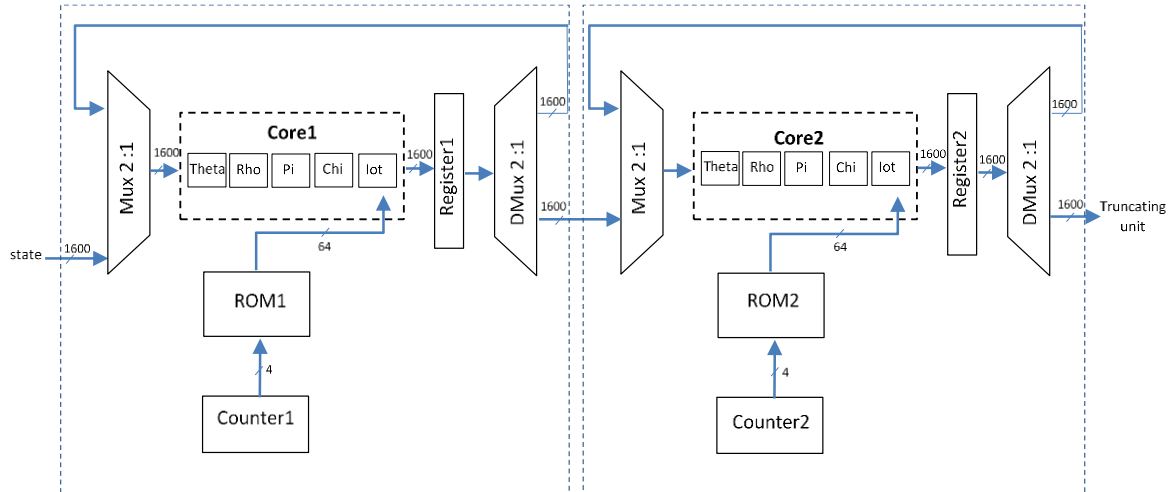


Figure 5. Pipelining two-stages

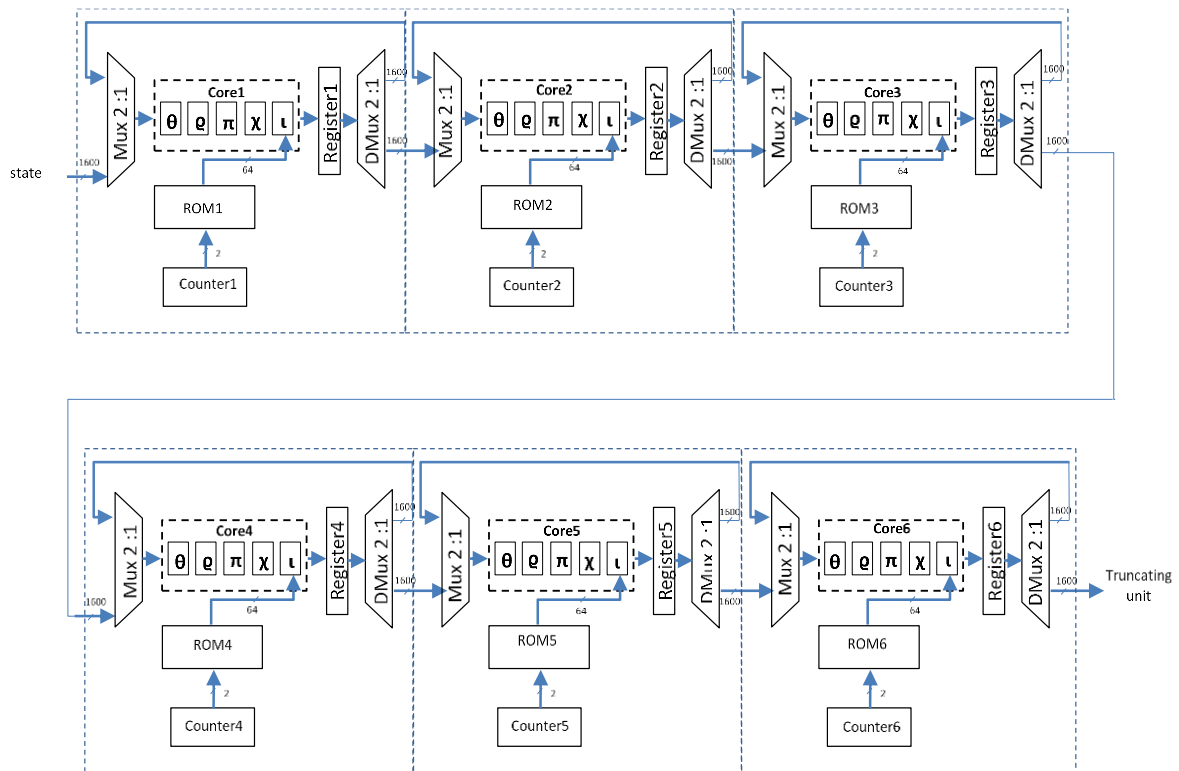


Figure 6. Pipelining Six-stages

4. RESULTS AND DISCUSSION

This section presents the results obtained in each case 2, 3, 4, and 6 pipelining stages. Also, comparisons between our results and the results of previous implementations already done for SHA-3 will be presented. The SHA-3 hardware architectures (for all cases) were coded in VHDL, synthesized, and implemented in V5 and V6 FPGA devices using the Xilinx ISE Design Suite v.12.1. Their correct functionality was, initially, verified through Post-Place and Route simulation via the ModelSim simulator.

The measurements were focused on the used FPGA Area resources, Frequency, design output Throughput, and Efficiency. Two FPGA boards were used: a Xilinx Virtex 5 XC5VLX110-1ff1760, and a Xilinx Virtex 6 XC6VLX760-1ff1760. Generally, there are three primary definitions of speed depending on the context of the problem: Throughput, Latency, and Timing [32].

The *Throughput* is obtained by using (2):

$$\text{Throughput} = \frac{\text{block size}(r) \times \text{frequency}}{\text{Latency}} \quad (2)$$

where *Latency* presents the number of clock cycles needed to compute the hash value.

The *Efficiency* is calculated by using (3).

$$\text{Efficiency} = \frac{\text{Throughput}}{\text{Area}} \quad (3)$$

The complete implementation results are given in Tables 2 and 3. Also, a comparison between the achieved results and the previous works is given in Table 4. For all the considered FPGA families, the *Throughput* increases about linearly with the number of the pipeline stages as depicted in Table 2. This is justified by the fact that the number of clock cycles is the dominant factor of the *Throughput* (2). The design reaches a maximum *Throughput* of 102.98 Gbps on Virtex 5 and 115.124 Gbps on Virtex 6 in the case of the 6 stages.

Table 2. Implementation results

FPGA devices	Stages	Latency	Maximum frequency (MHz)	Throughput (Gbps)			
				TP.224 (r=1125)	TP.256 (r=1088)	TP.384 (r=832)	TP.512 (r=576)
Virtex5	1	24	338.409	15.86	15.34	11.73	8.12
	2	12	338.40	31.72	30.68	23.46	16.24
	3	8	354.86	49.90	48.26	36.90	25.55
	4	6	354.86	66.54	64.35	49.21	34.07
	6	4	366.166	102.98	99.60	76.16	52.73
Virtex6	1	24	376.081	17.63	17.05	13.04	9.02
	2	12	393.10	36.85	35.64	27.25	18.87
	3	8	408.163	57.40	55.51	42.45	29.38
	4	6	426.257	79.92	77.29	59.11	40.92
	6	4	409.33	115.124	111.34	85.140	58.94

Table 3. Efficiency results

FPGA devices	stages	Latency	Area (Slices)	Efficiency (Mbps/Slices)			
				Efficiency 224	Efficiency 256	Efficiency 384	Efficiency 512
Virtex5	1	24	935	16.96	16.40	12.54	8.68
	2	12	2366	13.40	12.96	9.91	6.86
	3	8	2759	18.08	17.49	13.37	9.26
	4	6	4061	16.38	15.84	12.12	8.38
	6	4	6240	16.50	15.96	12.20	8.45
Virtex 6	1	24	1019	17.30	16.73	12.80	8.85
	2	12	2499	14.74	14.26	10.90	7.55
	3	8	3607	15.91	15.38	11.77	8.14
	4	6	4396	18.18	17.58	13.45	9.30
	6	4	6935	16.60	16.05	12.27	8.50

Table 4. Comparison of results on Virtex 5 and Virtex 6 for 512 bits output length

References	Area (slices)		Frequency (MHz)		Throughput (Gbps)		Efficiency (Mbps/slices)	
	V5	V6	V5	V6	V5	V6	V5	V6
[14]	4793	-	317.11	-	12.68	-	2.71	-
[11]	1702	1649	389	397	18.7	19.1	10.99	11.58
[12]	240	-	301.02	-	7.224	-	30.1	-
[33]	4361	5528	328.2	401.2	7.87	9.62	1.80	1.74
[34]	1388	1167	287.39	394.01	11.50	15.76	8.28	13.50
[13]	1647	1181	137.01	251.7	2.39	4.39	1.45	3.72
[35]	7224	10120	78.13	95.70	11.25	13.78	1.55	1.36
[17]	1192	-	223	-	5.35	-	4.49	-
[36]	2573	-	285	-	5.70	-	2.21	-
Our approach (6 stages)	6240	6935	366.166	409.33	52.72	58.94	8.44	8.49

On the other hand, due to the use of 6 cores, we notice that there is a significant increase in terms of *Area*, our design (Six-stage) requires 6240 slices on Virtex 5 and 6935 slices on Virtex 6. As can be noted, the *throughput* increases noticeably in Table 2, this is justified by the fact that the number of clock cycles is the dominant factor of the *throughput* equation as shown in (2). On the downside, the area occupied by the algorithm increase with the increase of the number of cores used in each one of the cases. The maximum *Frequency* and the *Throughput* are improved when moving to more modern FPGA families, the values of the *Frequency* and the *Throughput* obtained in the recent families are much better compared to the old families.

Table 4 shows a comparison between the proposed design with previous works in terms of *Area*, *Frequency*, *Throughput* (TP), and *Throughput* per area (TPS) or *Efficiency* for 512 bits output length, for FPGA implementation on Virtex 5 and Virtex 6. By comparing our implementation results with the results of the previous works, our contribution outperforms significantly in terms of *Throughput* 52.72 Gbps on Virtex 5 and 58.94 Gbps on Virtex 6 as shown in Table 4. On the downside, speed enhancement offered through pipelining comes at the expense of a large hardware area Cost 6240 slices.

5. CONCLUSION

In this paper pipelining architecture for the SHA-3 algorithm was proposed. We have implemented two, three, four, and six pipelining stages and studied the change of the performance values in each case. This work shows the impact of using the pipelining technique on SHA-3's performance. A noticeable enhancement of the *Throughput* with the increase of the stage's number, as in the case of the six-stage pipeline has the highest *Throughput*. On the other hand, this technique has an adversary effect on the area, which increases with the increase of the number of stages.




REFERENCES

- [1] X. Wang and H. Yu, "How to break MD5 and other hash functionsk MD5 and other hash functions," in *Lecture Notes in Computer Science*, vol. 3494, Springer Berlin Heidelberg, 2005, pp. 19–35. doi: 10.1007/11426639_2.
- [2] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3621, Springer Berlin Heidelberg, 2005, pp. 17–36. doi: 10.1007/11535218_2.
- [3] M. J. Dworkin, "SHA-3 standard: permutation-based hash and extendable-output functions," National Institute of Standards and Technology, Gaithersburg, MD, Jul. 2015. doi: 10.6028/NIST.FIPS.202.
- [4] R. Rivest, "The MD5 Message-Digest Algorithm." MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. [Online]. Available: <http://altronic-srl.com.ar/md5%20algorithm.pdf>
- [5] "SHA-1 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-1," 1995. [Online]. Available: <http://www.itl.ni.st.gov/fipspubs/fip180-1.htm>
- [6] "NIST: Federal information processing standard 180-2, secure hash standard," [Online]. Available: <https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2.pdf>
- [7] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Keccak sponge function family main document," *Nist*, pp. 1–121, 2010.
- [8] S. Neelima and R. Brindha, "512 bit-SHA3 design approach and implementation on field programmable gate arrays," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 8, no. 3, pp. 169–174, Nov. 2019, doi: 10.11591/ijres.v8.i3.pp169-174.
- [9] A. Sideris, T. Sanida, and M. Dasygenis, "High throughput implementation of the keccak hash function using the nios-II processor," *Technologies*, vol. 8, no. 1, Feb. 2020, doi: 10.3390/technologies8010015.
- [10] H. Mestiri, F. Kahri, B. Bouallegue, and M. Machhout, "An efficient hardware implementation of keccak algorithm," *International Journal of Scientific Engineering and Applied Science (IJSEAS)*, vol. 3, no. 12, 2017.
- [11] G. S. Athanasiou, G.-P. Makkas, and G. Theodoridis, "High throughput pipelined FPGA implementation of the new SHA-3 cryptographic hash algorithm," in *2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, May 2014, pp. 538–541. doi: 10.1109/ISCCSP.2014.6877931.
- [12] A. Arshad, D.-S. Kundi, and A. Aziz, "Compact implementation of SHA3-512 on FPGA," in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, Jun. 2014, pp. 29–33. doi: 10.1109/CIACS.2014.6861327.
- [13] T. Honda, H. Guntur, and A. Satoh, "FPGA implementation of new standard hash function Keccak," in *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, Oct. 2014, pp. 275–279. doi: 10.1109/GCCE.2014.7031105.
- [14] H. Mestiri, F. Kahri, M. Bedoui, B. Bouallegue, and M. Machhout, "High throughput pipelined hardware implementation of the KECCAK hash function," in *2016 International Symposium on Signal, Image, Video and Communications (ISIVC)*, 2016, pp. 282–286. doi: 10.1109/ISIVC.2016.7894001.
- [15] H. E. Michail, L. Ioannou, and A. G. Voyiatzis, "Pipelined SHA-3 implementations on FPGA," in *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, Jan. 2015, vol. 2015-Janua, pp. 13–18. doi: 10.1145/2694805.2694808.
- [16] A. Akin, A. Aysu, O. C. Ulusel, and E. Savaş, "Efficient hardware implementations of high throughput SHA-3 candidates keccak, luffa and blue midnight wish for single- and multi-message hashing," in *Proceedings of the 3rd international conference on Security of information and networks - SIN '10*, 2010, pp. 168–177. doi: 10.1145/1854099.1854135.
- [17] M. Sundal and R. Chaves, "Efficient FPGA implementation of the SHA-3 hash function," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2017, vol. 2017-July, pp. 86–91. doi: 10.1109/ISVLSI.2017.24.
- [18] M. Rao, T. Newe, I. Grout, and A. Mathur, "High speed implementation of a SHA-3 core on virtex-5 and virtex-6 FPGAs," *Journal of Circuits, Systems and Computers*, vol. 25, no. 7, Jul. 2016, doi: 10.1142/S0218126616500699.
- [19] T. N. Dat, K. Iwai, T. Matsubara, and T. Kurokawa, "Implementation of high speed hash function keccak on GPU," *International Journal of Networking and Computing*, vol. 9, no. 2, pp. 370–389, 2019, doi: 10.15803/ijnc.9.2_370.




- [20] B. Jungk and M. Stöttinger, "Serialized lightweight SHA-3 FPGA implementations," *Microprocessors and Microsystems*, vol. 71, Nov. 2019, doi: 10.1016/j.micpro.2019.102857.
- [21] H. E. Michail, G. S. Athanasios, V. Kelefouras, G. Theodoridis, and C. E. Goutis, "On the exploitation of a high-throughput SHA-256 FPGA design for HMAC," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 5, no. 1, pp. 1–28, Mar. 2012, doi: 10.1145/2133352.2133354.
- [22] P. Pessl and M. Hutter, "Pushing the limits of SHA-3 hardware implementations to fit on RFID," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8086, Springer Berlin Heidelberg, 2013, pp. 126–141. doi: 10.1007/978-3-642-40349-1_8.
- [23] M. Rao, T. Newe, I. Grout, and A. Mathur, "An FPGA-based reconfigurable IPSec AH core with efficient implementation of SHA-3 for high speed IoT applications," *Security and Communication Networks*, vol. 9, no. 16, pp. 3282–3295, Nov. 2016, doi: 10.1002/sec.1533.
- [24] M. Rao, T. Newe, and I. Grout, "Secure hash algorithm-3(SHA-3) implementation on xilinx FPGAs, suitable for IoT applications," *International Journal on Smart Sensing and Intelligent Systems*, vol. 7, no. 5, pp. 1–6, Feb. 2020, doi: 10.21307/ijssis-2019-018.
- [25] J. Rao, T. Ao, S. Xu, K. Dai, and X. Zou, "Design exploration of SHA-3 ASIP for IoT on a 32-bit RISC-V processor," *IEICE Transactions on Information and Systems*, no. 11, pp. 2698–2705, Nov. 2018, doi: 10.1587/transinf.2017ICP0019.
- [26] S. Ravikumar and D. Kavitha, "IoT based home monitoring system with secure data storage by keccak–chaotic sequence in cloud server," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 7, pp. 7475–7487, Jul. 2021, doi: 10.1007/s12652-020-02424-x.
- [27] X. Li, M. H. Ibrahim, S. Kumari, A. K. Sangaiah, V. Gupta, and K.-K. R. Choo, "Anonymous mutual authentication and key agreement scheme for wearable sensors in wireless body area networks," *Computer Networks*, vol. 129, pp. 429–443, Dec. 2017, doi: 10.1016/j.comnet.2017.03.013.
- [28] Y. Yang, D. He, N. Kumar, and S. Zeadally, "Compact hardware implementation of a SHA-3 core for wireless body sensor networks," *IEEE Access*, vol. 6, pp. 40128–40136, 2018, doi: 10.1109/ACCESS.2018.2855408.
- [29] B. Prathusha Laxmi and A. Chilambuchelvan, "GSR: geographic secured routing using SHA-3 algorithm for node and message authentication in wireless sensor networks," *Future Generation Computer Systems*, vol. 76, pp. 98–105, Nov. 2017, doi: 10.1016/j.future.2017.05.015.
- [30] M. Louk, H. Lim, and H. J. Lee, "Security system for healthcare data in cloud computing," *International Journal of Security and Its Applications*, vol. 8, no. 3, pp. 241–248, May 2014, doi: 10.14257/ijisa.2014.8.3.25.
- [31] P. Velmurugadass, S. Dhanasekaran, S. Shasi Anand, and V. Vasudevan, "Enhancing blockchain security in cloud computing with IoT environment using ECIES and cryptography hash algorithm," *Materials Today: Proceedings*, vol. 37, pp. 2653–2659, 2021, doi: 10.1016/j.matpr.2020.08.519.
- [32] S. Kilts, *Advanced FPGA design: architecture, implementation, and optimization*. John Wiley & Sons, Inc., 2007. doi: 10.1002/9780470127896.
- [33] A. Tragha, S. El Moumni, and M. Fettach, "High frequency implementation of cryptographic hash function Keccak-512 on FPGA devices," *International Journal of Information and Computer Security*, vol. 10, no. 4, 2018, doi: 10.1504/IJICS.2018.10016384.
- [34] F. Kahri, H. Mestiri, B. Bouallegue, and M. Machhout, "High speed FPGA implementation of cryptographic KECCAK hash function crypto-processor," *Journal of Circuits, Systems and Computers*, vol. 25, no. 4, Feb. 2016, doi: 10.1142/S0218126616500262.
- [35] S. El Moumni, M. Fettach, and A. Tragha, "High throughput implementation of SHA3 hash algorithm on field programmable gate array (FPGA)," *Microelectronics Journal*, vol. 93, Nov. 2019, doi: 10.1016/j.mejo.2019.104615.
- [36] G. Provelengios, P. Kitsos, N. Sklavos, and C. Koulamas, "FPGA-based design approaches of keccak hash function," in *2012 15th Euromicro Conference on Digital System Design*, Sep. 2012, pp. 648–653. doi: 10.1109/DSD.2012.63.

BIOGRAPHIES OF AUTHORS







Fatimazahraa Assad    Graduated from Hassan II University of Casablanca, in June 2015 with a master of Information Processing. She is currently a doctoral candidate in computer science at the Information Processing Laboratory. Her research interests concentrate on computer sciences, networks security, cryptographic hash functions, and hardware implementation on the field-programmable gate array. Her current project is the conception and implementation of cryptographic hash functions on the FPGA. She can be contacted at email: fatimazahraa.assad@gmail.com.







Mohamed Fettach    received his Doctorate of State degree (equivalent to Ph.D.) from Hassan II University, Casablanca, Morocco, in 2002. He is currently a Professor at the Faculty of Sciences Ben M'sik, Casablanca, Morocco. His research interests include the computer-aided design of electronic systems, logic design, and security of system information. He can be contacted at email: fettachmohamed@gmail.com.



Fadwa El Otmani     received her Master's degree in information processing from the Faculty of Sciences Ben M'Sick, University Hassan II of Casablanca, Morocco, in 2015. Since 2015, she has been preparing her Ph.D. in the field of automatic, renewable energies, and power electronic converters in the Faculty of Sciences Ben M'sick, University Hassan II of Casablanca, Morocco. The main topics of her research include renewable energy and nonlinear control techniques of high-gain power converter topologies. She can be contacted at email: fadwa.elotmani@gmail.com.



Abderrahim Tragha     received his degree in Applied Mathematics from the Mohammed V University, Morocco, 1983, and Doctorate of High Graduate Studies degree in the Theories of Computer Sciences from the Mohammed V University, Morocco, 1988 and Doctorate of State degree (or Ph.D.) July 2006 in Computer Sciences from the Hassan II Casablanca University, Morocco. He is currently a Professor in the Department of Mathematics and Computer Sciences. He is the Director of Information Treatment and Modeling Laboratory. His search major field is on systems engineering, in the security of system information, and computational linguistic. He can be contacted at email: atragha@yahoo.fr.